Effectiveness of cache pollution attacks in ICN cache services

A Thesis

presented to

the Faculty of the Department of Engineering Technology

University of Houston

In partial fulfillment

of the Requirements for the Degree

Master of Science

in Engineering Technology

By

Andia Karimipoor

Dec 2016

Acknowledgements

Firstly, I would like to express my special appreciation and thanks to my advisor Dr. Ricardo Lent for the continuous support of my study and related research, for his patience, motivation. His guidance helped me in all the time of research and writing of this thesis.

Besides my advisor, I would like to thank my thesis committee: Dr. Benhaddou and Dr. Yuan, for reviewing my work and providing their insightful comments and feedback.

Finally, I would like to thank my family and friends for their immense support throughout my degree program.

Abstract

Information-centric networking is a new technique for future Internet. The current Internet architecture was designed based on a host to host communication. In recent years there have been several efforts to replace the current IP-based Internet. The key idea of ICN is that the user will focus more on what exactly they want rather than from where to get the content. Different ICN architectures have developed. CCN (content centric networking), NDN (named data networks) and CDN (content delivery networks) are examples of ICN architectures. ICN has different structure than the host to host networks in terms of naming, routing, security and caching. All these new terms created the chance of new security threats and attacks on network. One of these security threats is possible attacks on ICN cache services.

In this Master thesis, we have studied cache pollution attacks on information centric networking and investigated the network performance by comparing the normal system to a system under cache pollution attacks. Delay and path length are the parameters that we have studied in both cases. However, we defined different caching sizes and policies to see the impact of attack, on small network versus large network and later, we extended our research by studying the impact of the attack on network when we have different attack and attack detection probabilities. It intends to tackle the challenges of security concerns on ICN cache services. The evolution of new network architecture raise great challenges to study security attacks on ICN. Therefore, we implemented an ICN architecture with python and we simulated the cache pollution attacks using FIFO and LRU caching policies to analyze the effectiveness of attack on different scale networks. We designed our large network topology inspired by Gnutella's networks which are considered large peer to peer networks.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and Motivations

Now a day's internet and transferring data through network is one of the most important part of peoples life. It is one of the greatest inventions of all time. It has made it possible for many people to do lots of things. People can go online shopping, transfer money, video chat with their friends and family, etc. Therefore, researchers has focused on having an internet architecture which is more efficient and faster. Rapid movement in internet architecture has changed the internet host to host communication system to new information centric networking model. This new architecture has made it possible to speed up the data transfer through the network by its unique design. The routers in ICN are able to cache those data which are requested before and are considered popular. When a user request for a file, the nearest source to the request, replies back to the user. Thus, this saves a lot of the time and is a way to better implement load balancing on our network. As the internet host to host communication is changing and replacing by ICN architectures, new security attacks have developed to weaken its performance. Thus, it is important to secure our network from such attacks.

As mentioned, every node in ICN is able to cache the requested content through the

path and if a new requested item is tagged as a popular and cache is full, routers use different caching policies such as LRU(least recently used), FIFO(first in first out) and LFU(least frequently used) to replace the content in cache. Cache pollution attack particularly targets the popular contents that are stored in cache and tries to fill the cache with unpopular data. This forces the users to get their requested content from origin source rather than the nearest location. There are two types of cache pollution attacks: locality disruption pollution and false-locality pollution [3]. In locality disruption pollution, attacker sends a large number of requests to weaken content locality in cache data by requesting unpopular data and in contrast, a false-Locality pollution attacker repeatedly requests a few unpopular objects to build false localities, thus misleading the system to cache these objects and waste precious cache space [4].

Zipf's law distribution is used in some cache services to store the content. Thus, in some research studies this distribution is used to simulate the ICN cache services [14-15]. However, in our implementation we used a python function to create random requests with exponential distribution to place the popular contents in cache which we will discuss it in detail, in the next sections.

This research work is mostly focusing on evaluating the effectiveness of cache pollution attacks and performing a performance analysis while we have false locality attacks. We are hopeful that our research results will help us to design a better detection and prevention mechanism in future.

1.2 Objective

Throughout this research study we are analyzing the performance of ICN cache services by implementing an ICN architecture in python. As mentioned in previous section ICN has improved the network performance in terms of speed and it is important to secure this architecture. Therefore, we decided to do a performance analysis and investigate the effectiveness of cache pollution attacks on ICN and we are hopeful that our results can be helpful in order to design a better detection mechanism in future.

1.3 Proposed approach

Firstly, to evaluate the caching performance of ICN, we implemented a generic ICN architecture with python and we calculated the delay and path length in a normal situation and later we added an attacker to the system to compare our results to our first experiment. We used LRU and FIFO cache replacement policies in our cache implementation and ran our experiment for each case separately. The primary idea came from the work which has been done in [6], but we added attack probabilities to analyze the performance with different detection probabilities. We did our experiment on a small scale network consists of three nodes and after that we repeated the same scenario on a larger scale network with 7 nodes as more than 5 network nodes is considered a large network based on Gnutella network model [19].

1.4 Contributions

This thesis mostly focuses on performing a performance analysis of ICN cache services on a large and small network topologies. We have implemented an ICN cache system with python and later we have simulated cache pollution attack on our system to run

our experiments. Major contribution to this system is the ability of evaluating system performance with and without cache pollution attacks while we have different attack possibilities attack detection probabilities, on the network.

1.5 Thesis organization

Chapter 2 will present the background basis of this thesis and an account of related work that provides an outline of various ICN architectures and recent studies around this area. We also have provided previous findings on ICN cache pollution attacks performance, the design methods they have used to implement the cache and some detection algorithms. In addition, we will describe other simulation tools which have been used before and will give a brief explanation about work studies related to LRU (least recently used) and FIFO (first in first out) cache replacement policies.

Chapter 3 will provide system architecture of our implementation. We will provide details about how our simulation tool works in normal and attack situations and how we are calculating the parameters (delay and path length). We will also explain separate designs of small and large networks that we have used to run our experiments.

Chapter 4 will provide evaluation and results of our experiment in each testbed for different attacks and different attack possibilities and detection probabilities. We also will include the results on a normal situation to do a comparison to find the effectiveness of attack in each cases.

Chapter 5 concludes the thesis and discusses possible future work in this area.

Chapter 2

Literature Review

In this section, we survey some of the relevant literature in the active research area of information centric networking and security concerns around ICN cache services.

Data object is one of the important components of ICN. A DO can be a document, file, audio, video, or streaming in ICN. In other words, the DO is the information that we access in the PC, laptop, and mobile device in Internet, and is independent of storage location [12]. ICN focuses on content retrieval from a network regardless of the storage location or physical representation of this content. This means that information is named, addressed and matched independently of its location, therefore it may be located anywhere in the network [1] A survey on Information centric networking describes core functionalities of ICN architectures [1]. ICN naming is based on three categories: hierarchical, self-certifying and attribute value pair based [2]. It is important to have a good naming method in order to increase the system performance [12]. The reason is that in ICN users request for the content by its name and without a unique naming method for each content, one name might be assigned to more than one content and this will cause retrieval of false content upon a request. ICN routing is based on two techniques: named resolution and named-based routing. In name resolution the content name is resolved into single or set of IP addresses and any shortest path routing can be used to route the content to the destination. Named-based routing uses content

names and stores content information all along the way and uses reverse path to send the content back to the requester. In the ICN architecture, as the user/network can use any available copy, security cannot bound to the end points or storage location. Therefore, new security concepts are required to be applied on content itself [2]. ICN caching enables caching the content on each node through the path. This has significantly affected user performance. It can also supply significant advantages for network operators and end users, such as lower the network load and faster response time for getting the content [11].

As mentioned earlier in previous sections, studies in [14-15] has focused on Zipf's law distribution formula to find the content that should be cached. For example, the pattern in requests shows that although there are thousands of links within a web, there are few that users request the most. Thousands of links within a web, there are few that users request the most. This pattern is so familiar for users who have studied Zipf's law distribution. Zipf's law states that given some corpus of natural language utterances, the frequency of any word is inversely proportional to its rank in the frequency table (i.e., the smaller the rank the higher the request frequency). Thus the most frequent word will occure approximately twice as often as the second most frequent word, three times as often as the third most frequent word [16]. To make it clear, if N is number of elements, k is their rank and s is the value of exponent characterizing the distribution, the population of N elements, the frequency of elements of rank k, f(k;s;N) based on Zipf's law is:

$$f(k;s,N) = \frac{1/k^s}{\sum_{n=1}^{N}(\frac{1}{n^s})}$$

As it is stated in [14], computer scientists have used this formula in designing the cache of content delivery networks and researches show that distribution of requests follow

Zipf's law distribution. Therefore, in designing our caching system, we used exponential distributed variables in a python function which is called expovariate, to generate random requests to fill the cache. Expovariate gives you random floating point numbers, exponentially distributed. In a Poisson process the size of the interval between consecutive events is exponential and its functionality is close to Zipf's law model in generating the numbers.

Different ICN architectures has been designed and developed in recent years. NDN (named data networks) and CCN (content centric networking) are two examples of new approached information centric networks. The key blocks of NDN are named data objects and NDN changes the semantics of network services from delivering the packet to a given destination address to fetching the object identified by a given name [7]. To enable this, NDN architectures provides the universal functionalities of routing, forwarding, and caching of named data objects on network nodes. More specifically, each data object has a URL-like name that is globally unique. When a router receives an object-requesting message (named interest in NDN) it queries its pending interest table (PIT) for the pending interests for the same data. If there are pending Interests, the newly arrived interest is aggregated with them; otherwise, the router looks up its Forwards Information Base (FIB) using the objects name and forwards the request to the appropriate interfaces toward the content servers that serve the data. The pending Interests are kept in the PIT until the corresponding data object traverses the node along the reverse path or times out. However, when a router has already cached the objects in its local content store, it can directly respond to the Interest with its cached content copy [4]. Same architecture works for CCN (content centric networking). First the publisher will broadcast its available contents to the network, and the routers will store the route to that specific content in their FIB and when a requester needs a content, it will send an interest packet to the network. As the routers have the capability of caching

the previously requested contents, it will first check its CS to see whether or not it has the data stored in the cache and if not it will forward the packet to the potential source depending on its FIB and store the interest packet information on it is PIT. After the publishers receives the request, it delivers the content to user on the reverse path [12]. Some studies has focused on the impact of cache pollution attacks on these two architectures [4] [5].

In [4], it focused on false locality pollution attacks in NDN. They first used their simulation tool to show that, with limited resources, a pollution attacker can cause considerable damage to NDN network. Second, they proposed an anti-pollution algorithm and used two methodologies based on probabilistic counting and bloom filter techniques to implement the algorithm on NDN backbone routers. Their simulation and results confirmed that their algorithm is efficient and successful in thawing a false-locality pollution attack.

The studies has also focused of different caching replacement strategies like LRU, LFU and FIFO to evaluate the performance of system during the attack. In high performance computer systems, in order to achieve a high-speed memory hierarchy, caching is considered to be the most popular way [27]. Cache controller responsibility is to decide which data should be removed from the cache for the new content when the cache is full. LRU is the least recently used content in the cache, while, LFU is least frequently used content which is based on how frequent a content has been requested. FiFO is designed based on first in first out algorithm and random replacement algorithm is designed based on replacing the items randomly in cache. The LRU is one of the most popular replacement policies. It mostly focus on reducing the chance of discarding information that will be needed soon. It records most popular data and replace those

data that has not been used for the longest time. In contrast, FIFO does not consider the popularity of the content, the first item in the cache is the first to be replaced with new item.

In [6] LRU and FIFO caching policies are simulated to analyze the bandwidth utilization and hit ratio of the ICN cache in normal situation and while cache pollution attack occurs. They have expand their experiments on large networks and considered that all nodes have the same cache size. Their results indicated that while the attack is moderately success in small networks, it potency decreased rapidly and it produced negligible impact in large network topology.

The work in [12] has also focus on LFU and FIFO caching policies but they most have focused on cache performance in normal situation with different cache replacement policies while requesting different type of applications. Their results demonstrate that LRU and FIFO have better performance than un-sequential content delivering and random under sequential. Also it shows that LRU and FIFO is a better fit for networking applications and cache hit ratio for both policies are the same when the experiments have same interest packet sending rate and time under sequential content delivering.

Different simulators has been proposed in [15] to examine cache performance and functionality. CCNx is a simulator for NDN and consists most of the operations included in CCN. It mostly focuses on changes in OSI layer architecture and pays less attention to transport and caching operations. To be able to cover these areas, new simulator software has been developed on top of ns-3 called ndnSIM. It is mostly based on ns-3 network simulator and provides all the requirements to evaluate the transport

behavior of networking in CCN. It uses separate C++ classes to model behavior of each network-layer entity in NDN: PIT (pending interest table) and FIB (forwarding information base), CS (content store), network and application interfaces, etc [15]. ndnSIM also uses different values of Zipf's parameter (α) for evaluating the performance of the network. CCNsim is another ICN simulator used in research studies and it is also written in C++ under the Omnet ++ and supports simulations of large catalogue and cache size.

Studies in [2] [23] [24] [18] have also discussed ICN cache services and cache pollution attacks security threats. They have also proposed some detection methods to deal with the attack. For example, in [2] they have proposed CacheShield to prevent cache pollution attacks and enhancing cache robustness. In addition, Zipf-like distribution is used for normal web requests and a uniform distribution is used for requesting unpopular data to simulate the attack. CacheShield identifies those objects that are requested frequently and it prepares a list of popular and unpopular objects. Thus, instead of caching every object, it first checks to see if the requested object is popular or not and if it passes this check the item will be cached. Therefore, it will keep unpopular data out of content store which greatly reduces the impact of cache pollution attacks. Their experiment results demonstrate that CacheShield maintains its robustness under different attack scenarios and even without attack hit ratio is higher compared to normal caching.

As mentioned before, in this thesis, we have implemented a generic ICN architecture using our own written simulator in python to evaluate the cache performance during cache pollution attacks like the work done in [6], but we have considered different attack possibilities and attack probabilities.

Chapter 3

System Architecture

In this chapter, we have explained the python implementation of our ICN architecture. We have described the functionality of our program and the topologies that we have used to do our performance analysis.

3.1. ICN application implementation

In our study, we have a python implementation of ICN cache system. Our implementation includes two separate configurations on client side and server sides. The program on the client side is designed to send requests for the files that are stored on servers. We have created a list of files in python program on client and we have used exponential distributed variables to randomly create request for the files. Fig 3.2 is displaying the histogram of our exponential distribution for user requests for a list of 50 files in our implementation. As it is shown in fig 3.2, file indexes between 0 to 10 and 10 to 20 in our list have higher probability of being requested than the files with indexes between 30 to 50. This will make the first half of the files to be requested the most and considered as popular data.

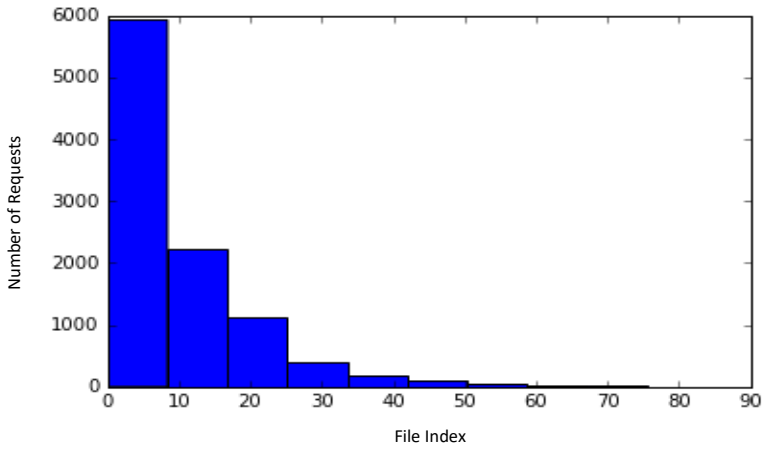

Figure: 3.2 User requests exponential distribution

In order to create the communication between client and servers, we used socket programming in our python implementation. Therefore, the servers will listen for the requests from the client and whenever the user sends a request for a file, server 1 will receive the request and will check its directory and its cache to see if it has the file or not. If it finds the file, it will send back the file to the user. Otherwise, first server will forward the request to the next server. The same process will happen in second server. However, if second server finds the file, it will send it back to the server 1. Server 1 will cache the content before sending it to the user. We considered a threshold for each file, based on the number of times that the item has been requested and we have designed a buffer as a temporary memory for the files that has been received from the next server and should be sent to the client, but they have not reached the required threshold to be stored in cache. This is for the time that we have a threshold bigger than 1. It is important to note that the server program is designed to run with different cache sizes and cache replacement policies. We should start the applications on servers by defining the cache size and replacement policy that we want to use to do our experiment. On the server side, our application is either able to use LRU cache replacement algorithm or FIFO and it is

also capable of running on as many servers as we want. In order to implement FIFO and LRU replacements policies two different classes on server side application are defined. These two classes are written based on FIFO and LRU algorithm and will be called when the cache size is full. Therefore when server receives a requests that it want to cache but its cache is full, it will use one of the mentioned cache replacement policies to replace the new content with an old one in cache.

We first start the program on servers by defining the cache size and cache replacement policy that we want to use for our experiment, so the server will start listening for a connection from client. Second we start the program on client and it will start sending requests for specific number of files that we have defined in our configuration.

In our study we first did our experiment using LRU cache replacement policy and again, repeated it for FIFO and in each case we created 10000 requests to compare their performances. And we continue to repeat our experiment for different cache sizes.

In the program we calculate the metrics that we want to use to do our measurements. In our experiment, delay (ms) and path length (hop count) are the two parameters that we have studied. Delay will be calculated between the time user sends a request till the time it receives the requested file and path length will count the number of hops that the request has passed to get to the file destination.

In the next section we will describe how we modified the client side program to simulate cache pollution attack.

3.2. Cache pollution attack implementation

In order to simulate cache pollution attacks, we modified the client side application by adding an attacker function to the program. The rest of the configuration is mostly similar to the time that we have just the client requesting for the files. The only difference is that the attacker function will request for unpopular data to fill the cache. Therefore, to make this happen, we again used the same random exponential distribution to create the attack requests but this time the function is configured to request mostly for the files between 40 to 50 rather than 0 to 20. We have also modified a part of the program to be able to run the attack requests with different possibilities and we called it pAttack. Thus, when we are creating 10000 requests, this part of the program will decide to send how many attack requests before or after the user requests. As a result, we have done our experiments for the time that we have just 1 % attack requests and 90 percent user requests on the system. Later, we repeated it for the time that we have 50% attack on the system to compare the results with the time that there was no attack on the system to measure the effectiveness of the attack in each case. We should configure the attack possibility before running the attack program.

We have also added different attack detection probabilities to the attack function, which will drop the attack traffic based on the given probability. We have used 0, 0.1, 0.25, 0.75 and 1 detection probabilities and we have called it dAttack. In our experiment we have repeated our experiment for each pAttack with different dAttack to evaluate for each attack possibilities which detection probability is the best to detect and stop the attack. dAttack should also configured before running the application.

Last but not least, same as the normal client program, we did repeat the whole set once

for LRU replacement policy and after that for FIFO and for different cache sizes to see the impact of attack on each policy and cache size.

In the next section we are going to explain the two different topologies we used to do our experiments.

3.3. Three node ICN topology

Figure: 3.1 Small network topology (3 nodes)

In order to do our analysis, we first configured our implementation on three Linux servers, Fig. 3.1. As it is mentioned in above figure, one node is defined as the client and two other nodes are configured as servers. As mentioned in previous section, we created 50 different files with various sizes on server "B" and we run the client program on client to generate requests for these 50 file with exponential distribution.

Server "A" is designed to be able to cache the contents that are not in cache when they are requested. As explained our program on server side is able to consider a threshold for each file, based on the number of times that the item has been requested. However, in our experiments, we decided to do all our experiments for the time that threshold is 1.

Thus, each file is cached on server "A" after being requested once.

In order to do our experiment using this small topology, First, we first started the server applications and we defined the cache replacement policy and the cache size we want to use and second we started the client program to start sending requests and receiving it from servers. We started the experiment with LRU replacement policy and cache size 1 and repeated the experiment for 5, 10, 15, 20 and 25 cache sizes and after that we repeated the whole set for FIFO.

Whenever the client sends a request for specific file name, server "A" will first receive the request. Next, it will search for the file on its local directories and then will look into its cache and replies back to the user if it finds the file. Otherwise, server "A" passes the request to server "B" and asks for the file. Same process will happen in server "B" for the requested file and it will send the file back to server "A" when it has the content stored in its cache or local directory. As the threshold is set "1", server "A" caches all contents received from server "B" and if the cache is full, it uses the replacement policy algorithm which is defined while running the application on server side, to replace the new item in cache. Moreover, server "A" uses its buffer to send the request back to the user after caching it. Our first experiment has been done on the topology and configuration explained above and we called it normal situation. And then we repeated the experiment on the same topology by running the attacker program. It is important to note that after each experiment we stored the results of average delay and average hop count to do our analysis.

3.3. Seven node ICN topology.

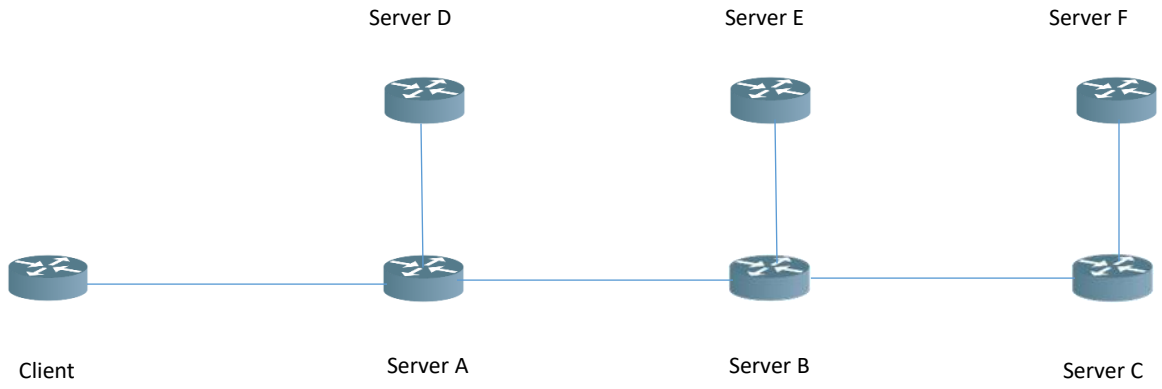

Figure: 3.3. Large network topology (7 nodes)

As our second experiment, and in order to extend our study on a larger scale network, we again implemented the same ICN architecture on 7 Linux servers, Fig. 3.2. The only differences in configuring the larger network are the distribution of the 50 files and the routing option which is added to servers. It should be noted that the reason we used this topology was to study the impact of attack on a network where the contents are more hops away from the client and we have more than one router through the path that is caching the data.

In this topology, instead of having all the files stored on the last server, we divided the files and distributed them on server D, server E and server F. The files were assigned completely randomly. We changed a part of application on server A, B and C, so they were able to route the requests to the destination based on the name of the requested file.

For example server A knows which files are stored on server D, and whenever a user sends a request for a file, it will first routed to server A. This server will first check to see if it has the file in its cache or not. If it finds the file it will send it back to the client. Otherwise, it will check the filename to see if it matches the files on server D or not. If it does, it will pass the request to server D, if it does not match, it will pass it to its next server which is server B. The same procedure will happen for other servers till the request gets to its destination.

After the topology set up, we again repeated the same experiments that we have done on small network topology and we collected delay and path length to do our analysis based on the results of both topologies. The number of samples that we used to do our test was again 10000 requests per each situation.

In the next section we will provide and review the results of our experiment for each situation and each topology.

Chapter 4

Results and Evaluation

4.1. Small network topology

In this section, we have discussed the results of our experiments on a small network topology. We have run our program for 10000 requests and calculated the average path length and delay for both attack and normal situations.

4.1.1. Delay (ms) results for LRU cache replacement policy with pAttack 0.5

Table 4.1 is showing the results of our experiment in normal situation and attack situation when pAttack is 0.5 with different attack detection probabilities. During this experiment, we are using LRU as our cache replacement policy and the experiment is done on a small network topology. The first column of the table is showing the values of delay (ms) while we do not have attack on the network and the other columns are showing the delay values (ms) during attack with attack detection probabilities of 0, 0.1, 0.25, 0.75, and 1.

As it is shown in fig. 4.1, first the attack has increased the delay and by repeating the experiment with different detection probabilities, the effect has been decreased and with dAttack 1 we have got close results to the normal situation. It is also obvious that as the cache size increases we have a considerable decrease in delay. That decrease also happens then there was no attack on system. The reason is that when we have more space to cache items, we can keep more contents in cache and this will increase the chance of getting the contents from the cache instead of from its origin. However, the variation amount between the attack and normal situation is higher in larger cache sizes. That means that the attack has been more effective on larger sizes as it can occupy more space of cache with attack requests.

Cache size	LRU Normal	LRU Attack(pdet0,pat0.5)	LRU Attack(pdet0.1,pat0.5)	LRU Attack(pdet0.25,pat0.5)	LRU Attack(pdet0.75,pat0.5)	LRU Attack(pdet1,pat0.5)
1	32.5394	34.3217	33.3027	33.3141	33.3364	32.6307
5	29.6703	31.1788	30.3755	30.1742	30.2820	29.5239
10	25.9192	30.0314	28.7989	28.6879	28.3980	25.4522
15	22.5593	27.9145	26.9171	26.2525	25.2503	22.6016
20	20.0527	24.5203	24.2346	24.0855	23.2799	20.6211
25	18.3352	23.1483	21.9959	21.9702	21.8460	18.5078

Table 4.1 Delay (ms) values of normal situation versus attack situation with LRU and pAttack 0.5

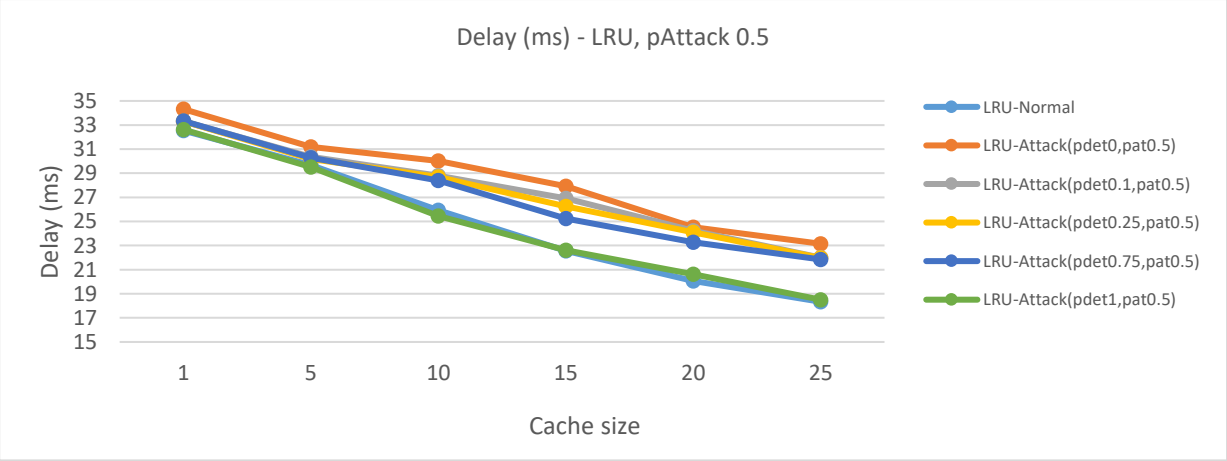

Figure: 4.1 Delay (ms) – LRU, pAttack 0.5

4.1.2. Delay (ms) results for FIFO cache replacement policy with pAttack 0.5

We repeated the previous experiment for FIFO cache replacement policy on a small network topology. Table 4.2 and fig 4.2 are showing the results of our experiment.

FIFO is performing a bit worse than LRU in normal situation, but the attack has affected LRU more than FIFO based on our collected results.

It is reasonable that LRU is acting better in normal situation than FIFO, as it will replace the least recently used content with the new requested one when cache is full. But this also has a reverse effect in attack situation as it will keep the attacks requests in cache longer than FIFO. Therefore, the attack was more successful on FIFO than LRU.

Same as LRU it is observed that by repeating the experiment for different dAttacks the performance is increasing but the best close results was the time that we had 100% detection probability. In addition, for the same reason that was explained for the same experiment with LRU policy, as the cache size has increased average delay value has decreased. But, this time although the variation has increased but compare to LRU, variation has not increased a lot during the attack as cache size increases. That is because the FIFO policy works based on first in, first out algorithm and will not keep the items in cache as long as LRU does, and thus, there is a more possibility that recent attack requests being replaced by newly user request and being removed from the cache.

Cache size	FIFO Normal	FIFO Attack(pdet0,pat0.5)	FIFO Attack(pdet0.1,pat0.5)	FIFO Attack(pdet0.25,pat0.5)	FIFO Attack(pdet0.75,pat0.5)	FIFO Attack(pdet1,pat0.5)
1	33.822	34.6115	33.8674	33.3251	33.7985	33.8790
5	29.8418	31.4483	31.3823	31.6095	31.2932	29.9509
10	26.3845	29.0828	29.0992	29.0166	28.4436	26.4747
15	23.4383	26.4658	26.1851	26.0099	26.0899	23.1800
20	21.2039	24.7192	24.0952	24.1470	24.1007	21.5016
25	20.4745	22.0211	22.4007	22.0527	22.0090	19.9780

Table 4.2 Delay (ms) values of normal situation versus attack situation with FIFO and pAttack 0.5

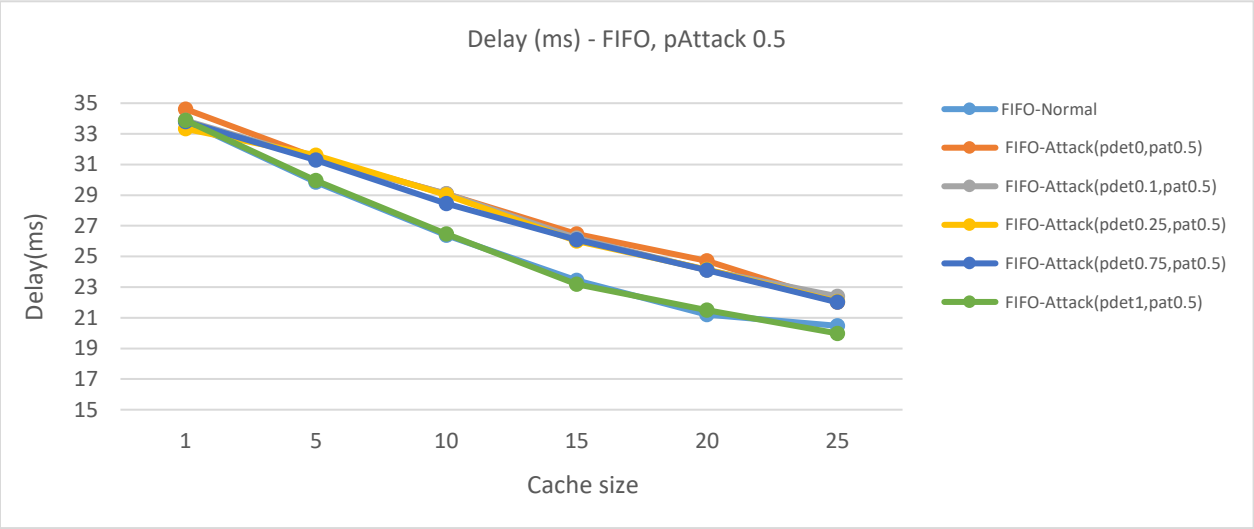

Figure: 4.2 Delay (ms) – FIFO, pAttack 0.5

4.1.3. Delay (ms) results for LRU cache replacement policy with pAttack 0.1

After running the experiments with pAttack 0.5, we changed the configuration of attacker application on client side to create attack requests with 0.1 possibility. We did the experiment for the same set of detection probabilities and as it is shown in table 4.3 and fig. 4.3, this time the effect of the attack was less than the time we had 0.5 attack possibility. This displays that with more attack requests the attack is more effective.

Cache size	LRU Normal	LRU Attack(pdet0,pat0.5)	LRU Attack(pdet0.1,pat0.5)	LRU Attack(pdet0.25,pat0.5)	LRU Attack(pdet0.75,pat0.5)	LRU Attack(pdet1,pat0.5)
1	32.5394	34.0227	32.9141	32.5700	32.5655	32.2903
5	29.6703	30.9394	29.9065	29.7577	29.5857	28.9480
10	25.9192	28.5779	26.6040	26.3160	26.4869	25.9331
15	22.5593	24.9685	24.5641	24.1744	23.2927	22.8484
20	20.0527	22.7061	22.6195	22.4869	21.2222	20.3309
25	18.3352	21.0119	21.0076	21.1542	20.0299	19.0045

Table 4.3 Delay (ms) values of normal situation versus attack situation with LRU and pAttack 0.1

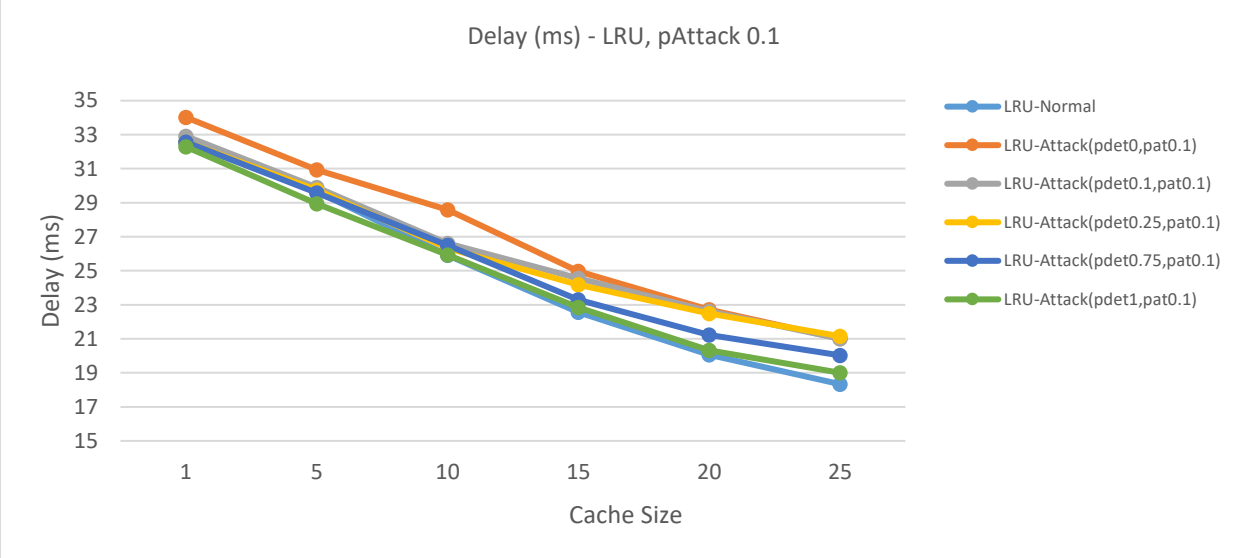

Figure: 4.3 Delay (ms) – LRU, pAttack 0.1

4.1.4. Delay (ms) results for FIFO cache replacement policy with pAttack 0.1

The experiment has also been repeated for FIFO cache replacement policy. Table 4.4 and fig. 4.4 are showing the values of delay while pAttack is 0.1. We can again say that the effect of the attack on FIFO is less than LRU.

Cache size	FIFO Normal	FIFO Attack(pdet0,pat0.5)	FIFO Attack(pdet0.1,pat0.5)	FIFO Attack(pdet0.25,pat0.5)	FIFO Attack(pdet0.75,pat0.5)	FIFO Attack(pdet1,pat0.5)
1	33.822	33.93	33.9536	33.9646	33.8568	33.7975
5	29.8418	31.9671	31.6778	30.0715	29.9820	29.9548
10	26.3845	28.9038	28.3301	27.5528	27.3202	26.4107
15	23.4383	25.9674	25.8102	25.0446	24.2755	23.6185
20	21.2039	22.3863	22.1542	22.0090	21.7530	21.4159
25	20.4745	20.9378	20.7547	20.5065	20.4727	20.3695

Table 4.4 Delay (ms) values of normal situation versus attack situation with FIFO and pAttack 0.1

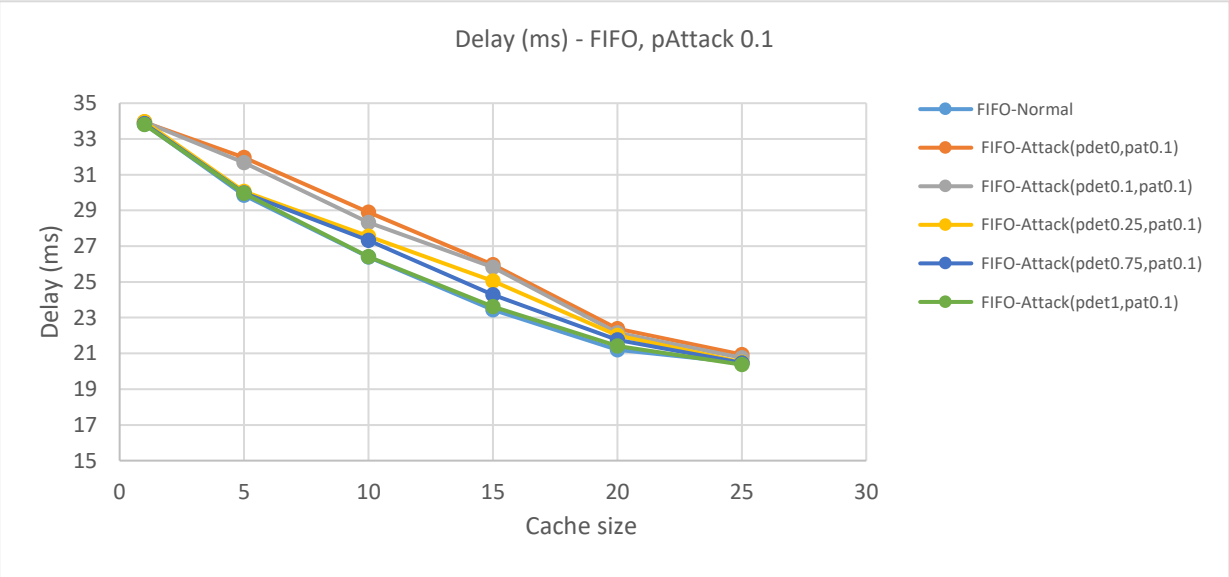

Figure: 4.4 Delay (ms) – LRU, pAttack 0.1

4.1.5. Path length (hop count) results for LRU cache replacement policy with pAttack 0.5

Path length was the second parameter that we studied during our performance analysis experiments and as there are several parameters that might affect the results of delay values, for instance, any network problem or any additional application running on the servers or client might affect the average delay values. Therefore, path length results seemed to be more accurate. Table 4.5 and fig 4.5 are showing the results while pAttack

was 0.5 and we were using LRU cache replacement policy. We can again say that like the results we got from average delay, as the cache size increases we have a considerable decrease in path length. It is important to note that in order to count the hops we started from 0, which means that if the content is located in the first hop, path length will be set to 0 and if it is in the second hop, it will be set to 1. Therefore, in a small topology the path length is either 0 or 1 as we just have 2 hops.

Cache size	LRU Normal	LRU Attack(pdet0,pat0.5)	LRU Attack(pdet0.1,pat0.5)	LRU Attack(pdet0.25,pat0.5)	LRU Attack(pdet0.75,pat0.5)	LRU Attack(pdet1,pat0.5)
1	0.9487	0.9727	0.9707	0.9697	0.9609	0.9505
5	0.759	0.8628	0.8548	0.8514	0.8026	0.7531
10	0.5458	0.7350	0.7195	0.7140	0.6272	0.5349
15	0.3638	0.6204	0.6056	0.6028	0.4735	0.3716
20	0.2337	0.5080	0.4839	0.4675	0.3598	0.2278
25	0.1334	0.3999	0.3894	0.3586	0.2538	0.1331

Table 4.5 Path length (hop count) values of normal situation versus attack situation with LRU and pAttack 0.5

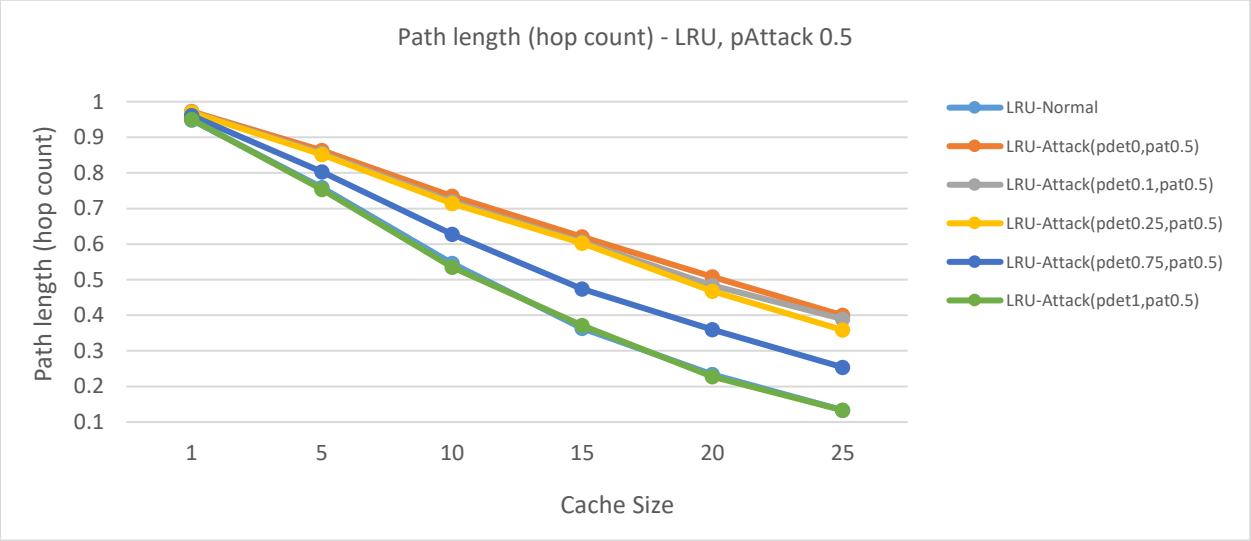

Figure: 4.5 Path length (hop count) – LRU, pAttack 0.5

4.1.6. Path length (hop count) results for FIFO cache replacement policy with pAttack 0.5

We repeated the previous experiment with FIFO cache replacement policy. Table 4.6 and fig. 4.6 are showing the results of our experiment.

Cache size	FIFO Normal	FIFO Attack(pdet0,pat0.5)	FIFO Attack(pdet0.1,pat0.5)	FIFO Attack(pdet0.25,pat0.5)	FIFO Attack(pdet0.75,pat0.5)	FIFO Attack(pdet1,pat0.5)
1	0.9505	0.9715	0.9722	0.9720	0.9566	0.9449
5	0.7665	0.8622	0.8600	0.8507	0.8045	0.7676
10	0.5703	0.7497	0.7425	0.7143	0.6490	0.5638
15	0.4019	0.6336	0.6286	0.6089	0.5221	0.4215
20	0.3072	0.5265	0.5141	0.4950	0.4031	0.2881
25	0.1993	0.4198	0.4132	0.3877	0.3148	0.2084

Table 4.6 Path length (hop count) values of normal situation versus attack situation with FIFO and pAttack 0.5

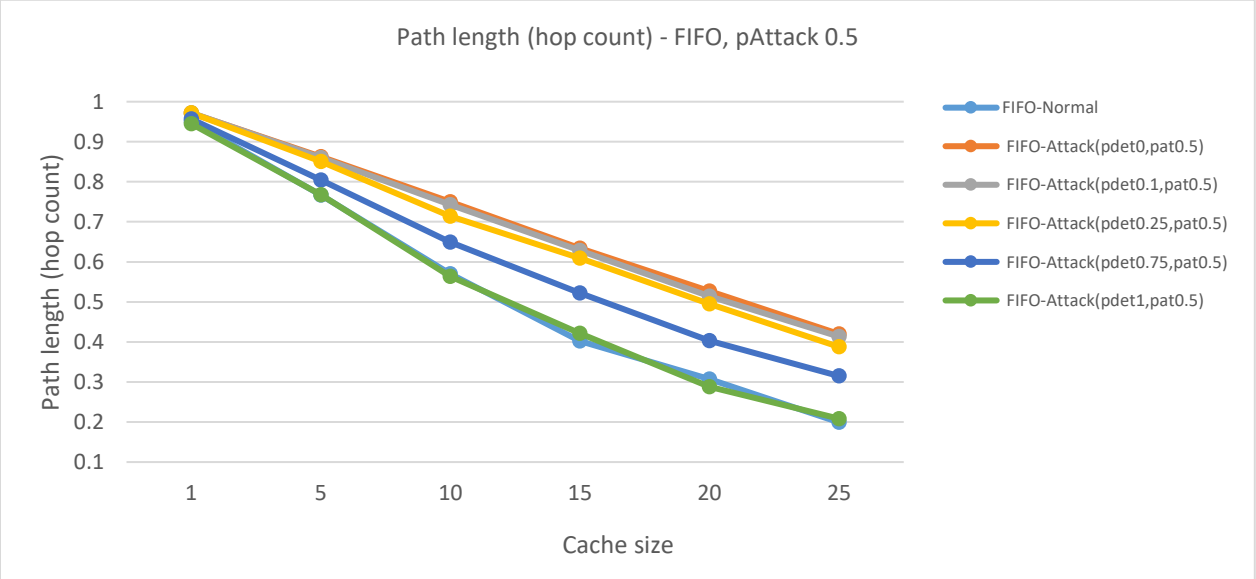

Figure: 4.6 Path length (hop count) – FIFO, pAttack 0.5

4.1.7. Path length (hop count) results for LRU cache replacement policy with pAttack 0.1

Below are the results of average path length with pAttack 0.1 and LRU cache replacement policy.

Cache size	LRU Normal	LRU Attack(pdet0,pat0.5)	LRU Attack(pdet0.1,pat0.5)	LRU Attack(pdet0.25,pat0.5)	LRU Attack(pdet0.75,pat0.5)	LRU Attack(pdet1,pat0.5)
1	0.9487	0.9523	0.9519	0.9524	0.9526	0.9437
5	0.759	0.7768	0.7712	0.7680	0.7607	0.76
10	0.5458	0.5856	0.5790	0.5829	0.5494	0.5515
15	0.3638	0.4350	0.4206	0.4187	0.3720	0.361
20	0.2337	0.2957	0.2803	0.2802	0.2497	0.227
25	0.1334	0.1970	0.1909	0.1809	0.1624	0.138

Table 4.7 Path length (hop count) values of normal situation versus attack situation with LRU and pAttack 0.1

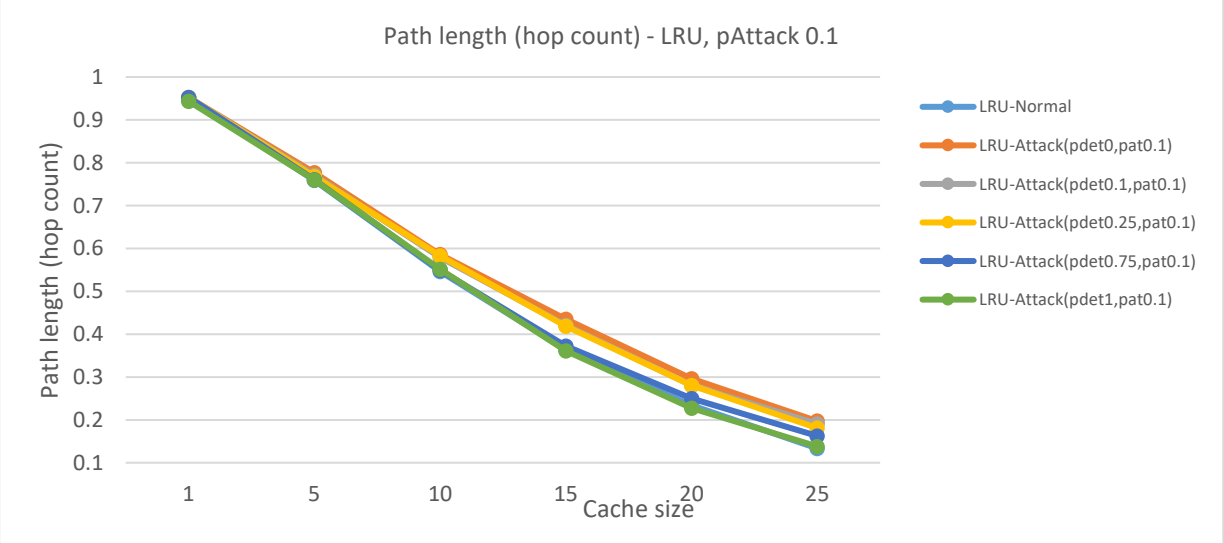

Figure: 4.7 Path length (hop count) – LRU, pAttack 0.1

4.1.8. Path length (hop count) results for FIFO cache replacement policy with pAttack 0.1

Table 4.8 includes the results of our experiment when pAttack is 0.1 and cache replacement policy is FIFO.

Cache size	FIFO Normal	FIFO Attack(pdet0,pat0.5)	FIFO Attack(pdet0.1,pat0.5)	FIFO Attack(pdet0.25,pat0.5)	FIFO Attack(pdet0.75,pat0.5)	FIFO Attack(pdet1,pat0.5)
1	0.9505	0.9582	0.9518	0.9510	0.9532	0.9484
5	0.7665	0.7893	0.7956	0.7796	0.7735	0.7606
10	0.5703	0.6153	0.6127	0.6024	0.5841	0.5761
15	0.4019	0.4793	0.4780	0.4753	0.4319	0.4064
20	0.3072	0.3626	0.3663	0.3512	0.3342	0.294
25	0.1993	0.2654	0.2657	0.2616	0.2303	0.2026

Table 4.8 Path length (hop count) values of normal situation versus attack situation with FIFO and pAttack 0.1

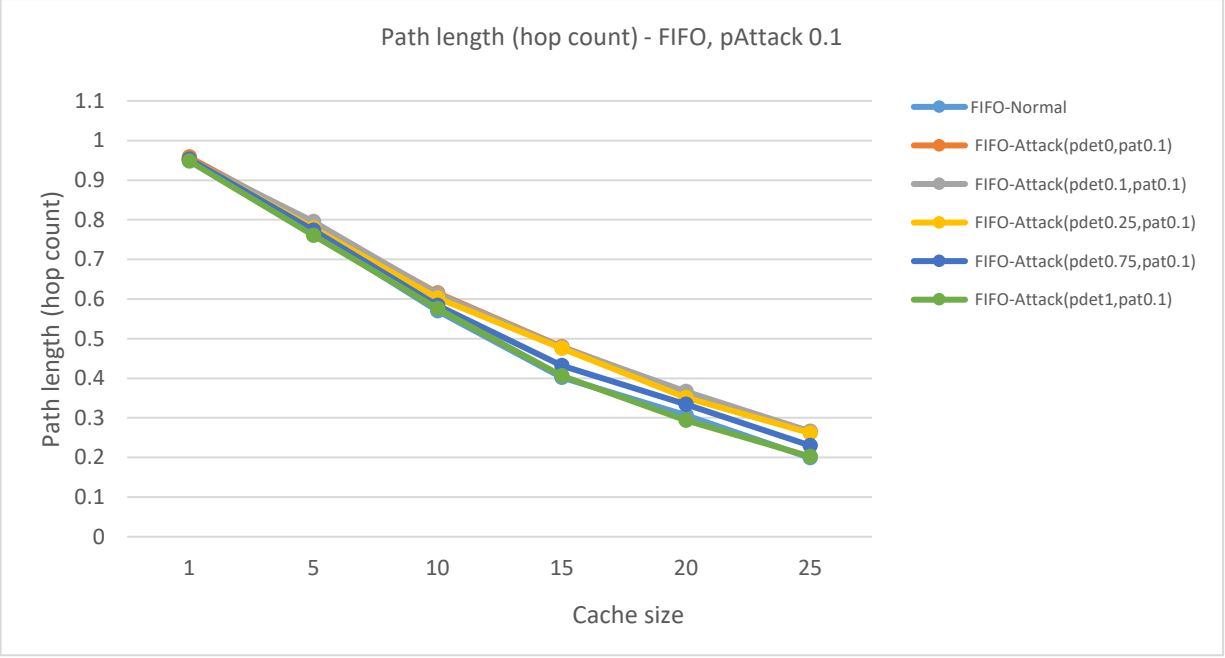

Figure: 4.8 Path length (hop count) – FIFO, pAttack 0.1

4.2. Large network topology

We repeated all the experiments on our 7 node topology and below are the results for each case.

4.2.1. Delay (ms) results for LRU cache replacement policy with pAttack 0.5

Table 4.9 is displaying the results of our experiments for pAttack 0.5 and LRU cache replacement policy on large network topology.

Cache size	LRU Normal	LRU Attack(pdet0,pat0.5)	LRU Attack(pdet0.1,pat0.5)	LRU Attack(pdet0.25,pat0.5)	LRU Attack(pdet0.75,pat0.5)	LRU Attack(pdet1,pat0.5)
1	48.6803	50.8563	50.0904	50.0120	50.0322	49.0133
5	39.447	45.7166	45.2104	44.0306	43.9014	39.0914
10	31.7257	40.5350	39.6265	38.8791	35.6871	32.1033
15	25.8096	34.7375	32.1747	31.7664	30.1429	26.0128
20	22.47	30.9456	27.1384	25.1041	24.7049	23.4957
25	19.9257	28.2328	26.5419	24.002	22.6773	20.9280

Table 4.9 Delay (ms) values of normal situation versus attack situation with LRU and pAttack 0.5

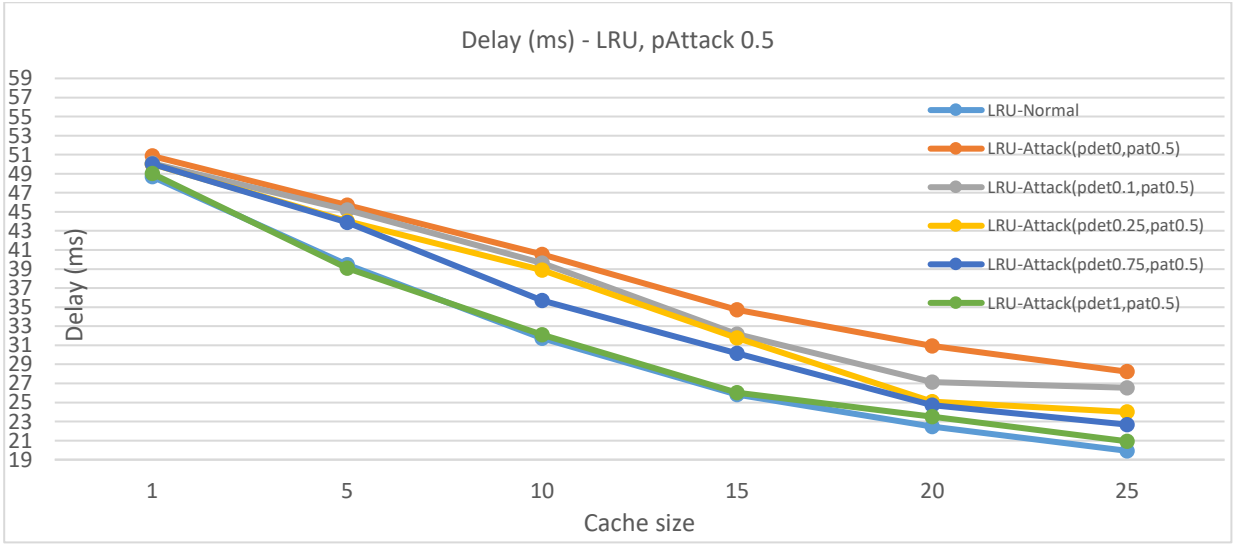

Figure: 4.9 Delay (ms) – LRU, pAttack 0.5

4.2.2. Delay (ms) results for FIFO cache replacement policy with pAttack 0.5

Table 4.10 is displaying the results of our experiments for pAttack 0.5 and FIFO cache replacement policy on large network topology.

Cache size	FIFO Normal	FIFO Attack(pdet0,pat0.5)	FIFO Attack(pdet0.1,pat0.5)	FIFO Attack(pdet0.25,pat0.5)	FIFO Attack(pdet0.75,pat0.5)	FIFO Attack(pdet1,pat0.5)
1	50.1689	52.6376	51.4721	50.2024	50.1916	50.0187
5	42.6464	44.5635	44.1700	43.9786	43.3581	41.9053
10	33.6546	39.9829	38.0456	37.4324	34.4716	33.8418
15	29.2196	32.8632	31.5105	31.2019	30.5109	29.5102
20	25.0308	30.8736	29.5217	29.1757	28.0464	25.8153
25	21.7517	27.4056	26.5912	25.9794	24.8135	22.1853

Table 4.10 Delay (ms) values of normal situation versus attack situation with FIFO and pAttack 0.5

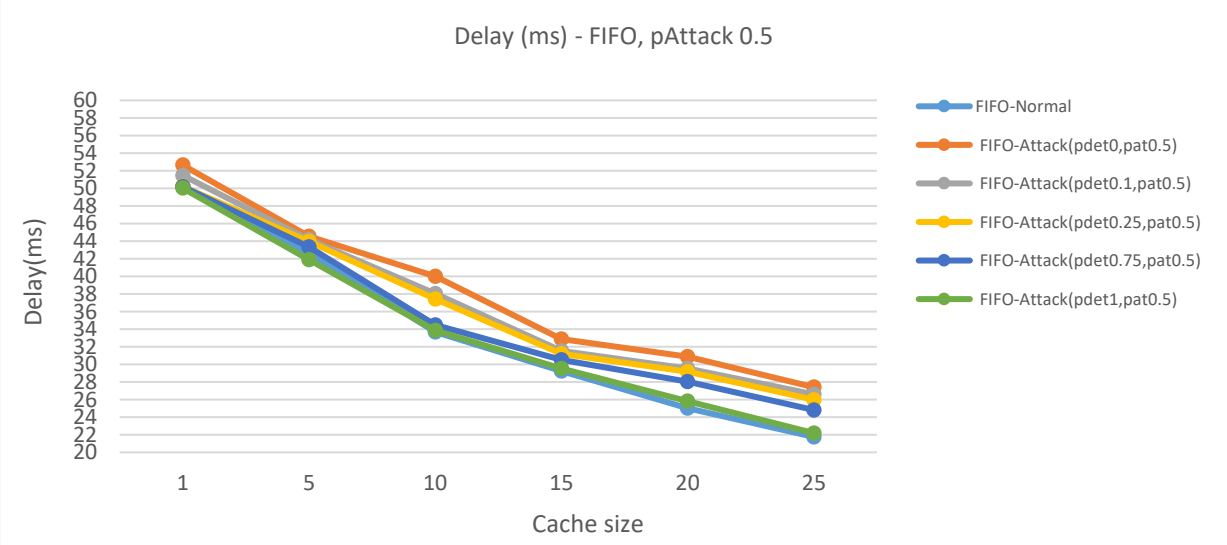

Figure: 4.10 Delay (ms) – FIFO, pAttack 0.5

4.2.3. Delay (ms) results for LRU cache replacement policy with pAttack 0.1

Table 4.11 is displaying the results of our experiments for pAttack 0.1 and LRU cache replacement policy on large network topology.

Cache size	LRU Normal	LRU Attack(pdet0,pat0.5)	LRU Attack(pdet0.1,pat0.5)	LRU Attack(pdet0.25,pat0.5)	LRU Attack(pdet0.75,pat0.5)	LRU Attack(pdet1,pat0.5)
1	48.6803	50.1561	49.8132	49.7029	49.1141	48.9413
5	39.447	43.1373	42.5607	41.9142	41.0678	39.8149
10	31.7257	39.2726	38.9026	36.1175	34.4498	31.8985
15	25.8096	32.4409	31.8232	30.7849	28.3223	26.9859
20	22.47	28.0702	26.5553	24.0199	23.4245	22.9871
25	19.9257	24.3466	23.6507	22.1902	21.0419	20.9348

Table 4.11 Delay (ms) values of normal situation versus attack situation with LRU and pAttack 0.1

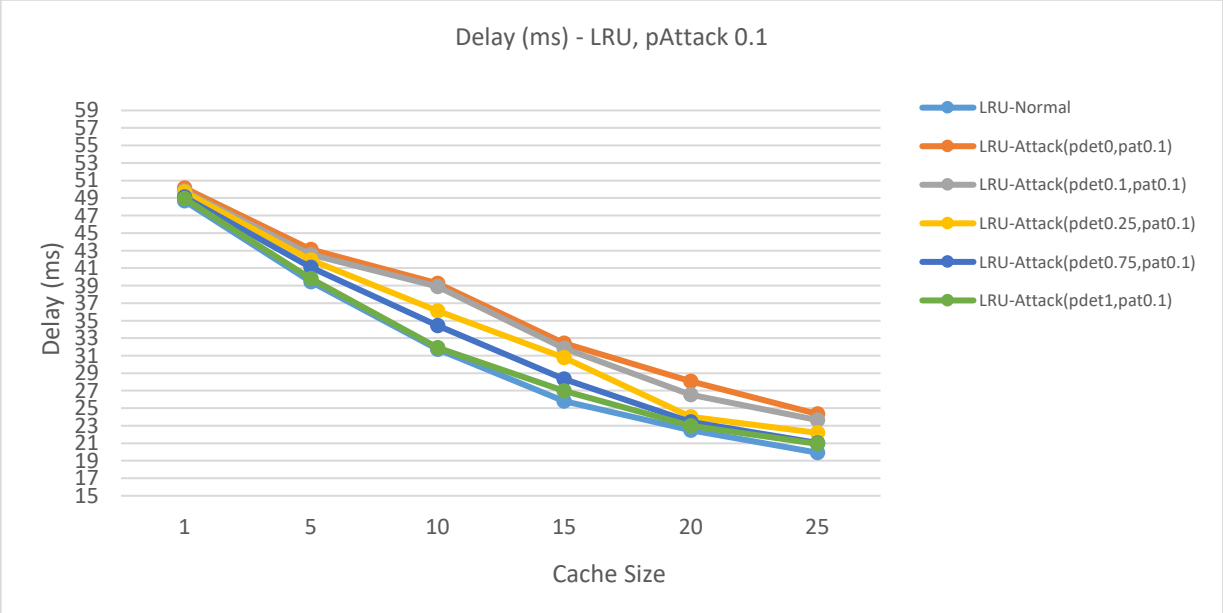

Figure: 4.11 Delay (ms) – LRU, pAttack 0.1

4.2.4. Delay (ms) results for FIFO cache replacement policy with pAttack 0.1

Table 4.12 is displaying the results of our experiments for pAttack 0.1 and FIFO cache replacement policy on large network topology.

Cache size	FIFO Normal	FIFO Attack(pdet0,pat0.5)	FIFO Attack(pdet0.1,pat0.5)	FIFO Attack(pdet0.25,pat0.5)	FIFO Attack(pdet0.75,pat0.5)	FIFO Attack(pdet1,pat0.5)
1	50.1689	51.9949	51.0854	50.0939	50.5511	50.2145
5	42.6464	44.1653	43.6482	43.0823	42.9295	42.5604
10	33.6546	38.5166	37.0719	35.0926	33.9375	33.4111
15	29.2196	31.9283	31.0575	31.0992	30.6619	29.1045
20	25.0308	29.8719	29.0114	27.7393	26.7491	25.2479
25	21.7517	25.9994	24.7812	23.4695	22.6509	21.9917

Table 4.12 Delay (ms) values of normal situation versus attack situation with FIFO and pAttack 0.1

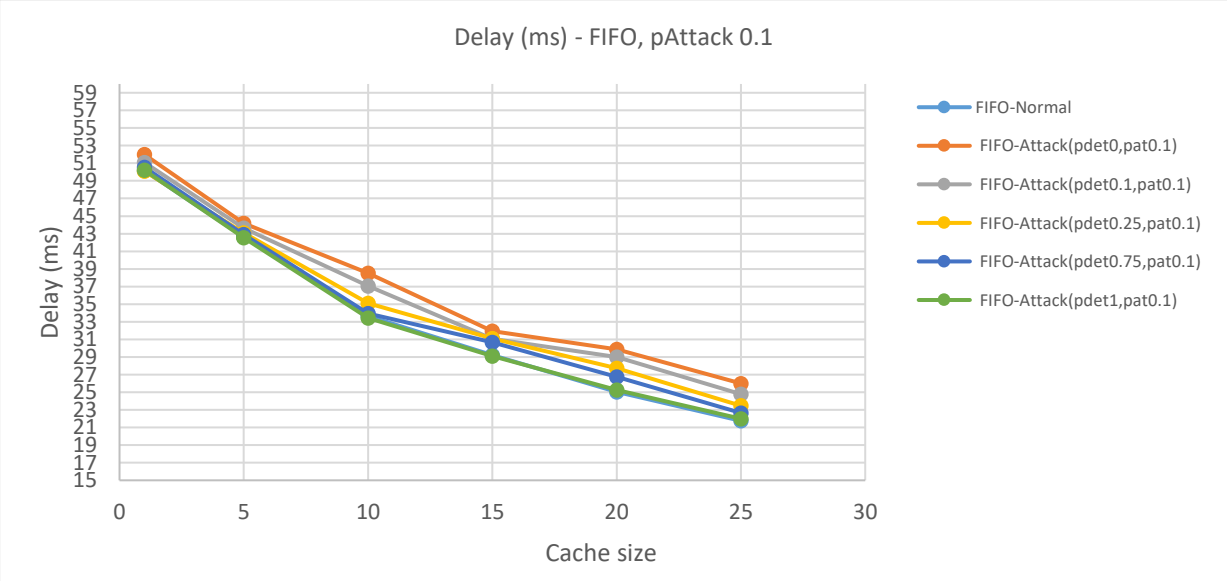

Figure: 4.12 Delay (ms) – FIFO, pAttack 0.1

4.2.5. Path length (hop count) results for LRU cache replacement policy with pAttack 0.5

Table 4.13 is displaying the results of average path length for pAttack 0.5 and LRU cache replacement policy on large network topology. It is important to note that, just as same as small network topology, we again counted the hops from 0.

Cache size	LRU Normal	LRU Attack(pdet0,pat0.5)	LRU Attack(pdet0.1,pat0.5)	LRU Attack(pdet0.25,pat0.5)	LRU Attack(pdet0.75,pat0.5)	LRU Attack(pdet1,pat0.5)
1	1.772	1.8568	1.8312	1.8488	1.8118	1.7872
5	1.299	1.5853	1.5545	1.5368	1.4050	1.2784
10	0.805	1.2239	1.2081	1.15329	0.9916	0.7739
15	0.4834	0.9155	0.9119	0.8652	0.6704	0.4817
20	0.283	0.6731	0.6748	0.6263	0.4722	0.2795
25	0.1589	0.4883	0.4804	0.4392	0.3161	0.1636

Table 4.13 Path length (hop count) values of normal situation versus attack situation with LRU and pAttack 0.5

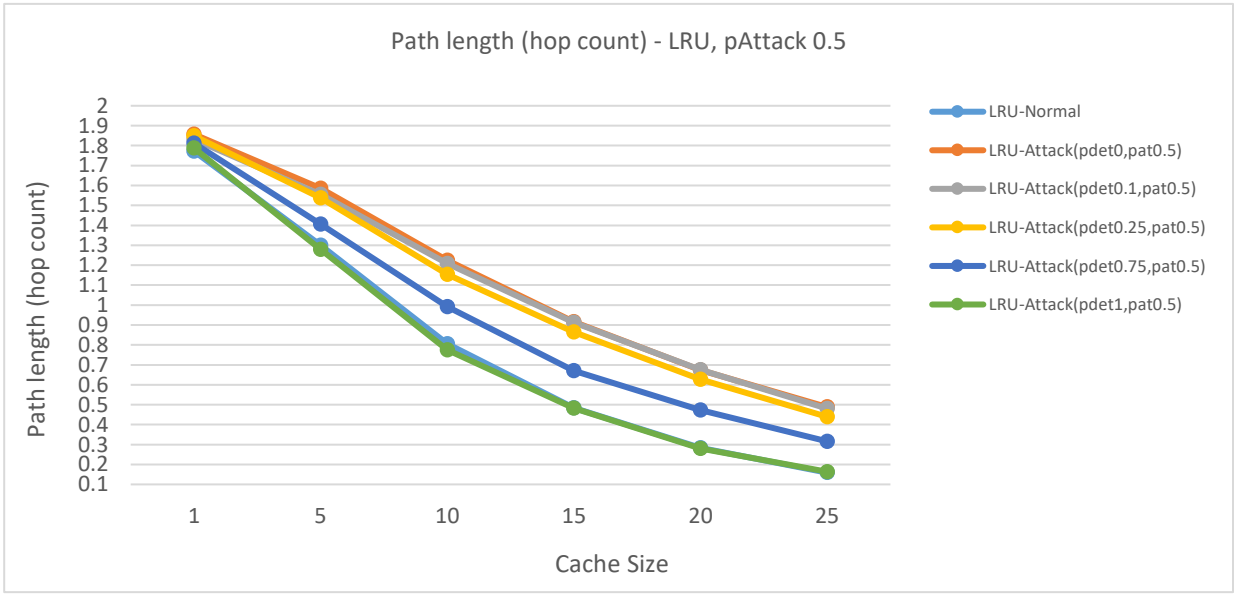

Figure: 4.13 Path length (hop count) – LRU, pAttack 0.5

4.2.6. Path length (hop count) results for FIFO cache replacement policy with pAttack 0.5

Table 4.14 is displaying the results of average path length for pAttack 0.5 and FIFO cache replacement policy on large network topology.

Cache size	FIFO Normal	FIFO Attack(pdet0,pat0.5)	FIFO Attack(pdet0.1,pat0.5)	FIFO Attack(pdet0.25,pat0.5)	FIFO Attack(pdet0.75,pat0.5)	FIFO Attack(pdet1,pat0.5)
1	1.7821	1.8350	1.8426	1.8542	1.8160	1.7887
5	1.2982	1.5554	1.5562	1.5070	1.4092	1.2793
10	0.8485	1.2072	1.1846	1.1585	1.0220	0.8307
15	0.5616	0.9140	0.9197	0.8791	0.7400	0.5565
20	0.3613	0.6814	0.6915	0.6590	0.5308	0.3709
25	0.2372	0.5229	0.5128	0.4855	0.3811	0.2365

Table 4.14 Path (hop count) values of normal situation versus attack situation with FIFO and pAttack 0.5

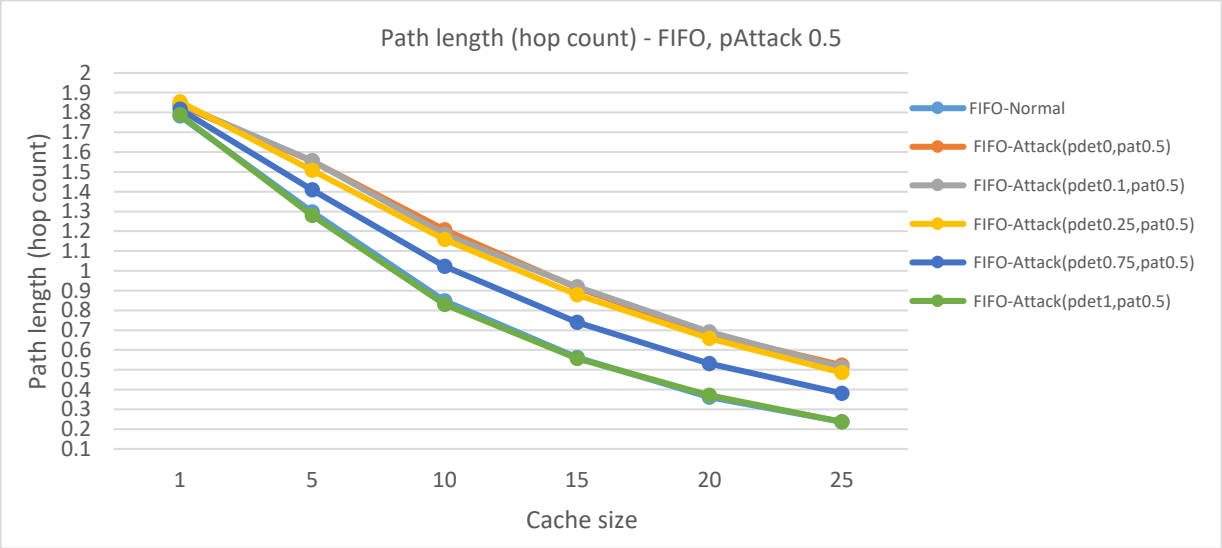

Figure: 4.14 Path length (hop count) – FIFO, pAttack 0.5

4.2.7. Path length (hop count) results for LRU cache replacement policy with pAttack 0.1

Table 4.15 is displaying the results of average path length for pAttack 0.1 and LRU cache replacement policy on large network topology.

Cache size	LRU Normal	LRU Attack(pdet0,pat0.5)	LRU Attack(pdet0.1,pat0.5)	LRU Attack(pdet0.25,pat0.5)	LRU Attack(pdet0.75,pat0.5)	LRU Attack(pdet1,pat0.5)
1	1.772	1.8170	1.8004	1.8086	1.7874	1.7934
5	1.299	1.3407	1.3223	1.3149	1.2736	1.2832
10	0.805	0.8992	0.8890	0.8765	0.8195	0.8082
15	0.4834	0.5907	0.5847	0.5804	0.4920	0.4754
20	0.283	0.3705	0.3782	0.3779	0.3115	0.2805
25	0.1589	0.2477	0.2400	0.2257	0.1804	0.1496

Table 4.15 Path length (hop count) values of normal situation versus attack situation with LRU and pAttack 0.1

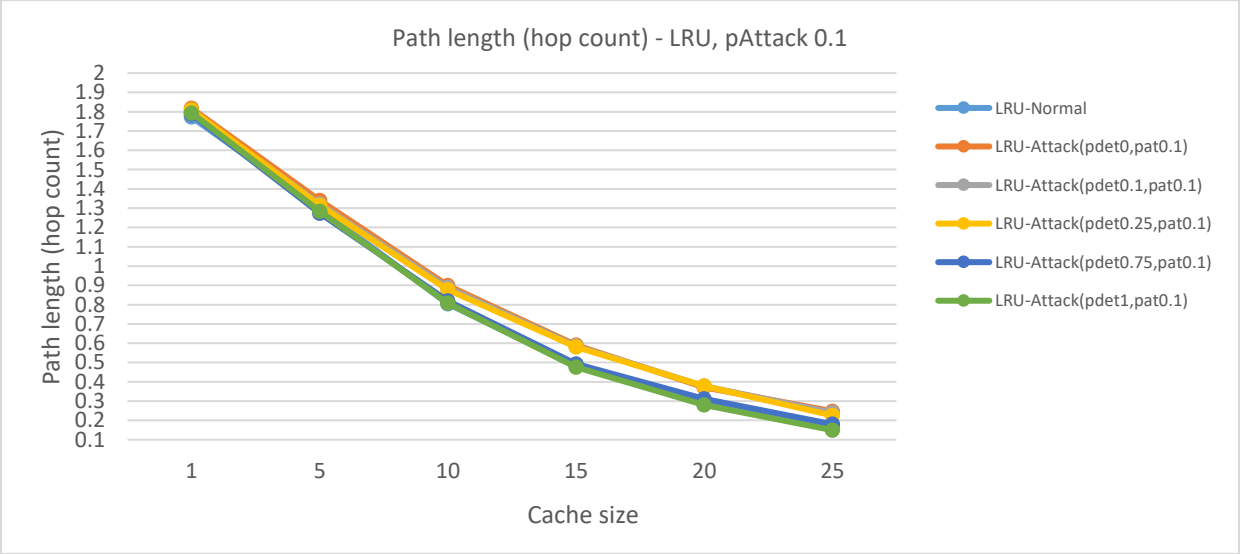

Figure: 4.15 Path length (hop count) – LRU, pAttack 0.1

4.2.8. Path length (hop count) results for FIFO cache replacement policy with pAttack 0.1

Table 4.16 is displaying the results of average path length for pAttack 0.1 and FIFO cache replacement policy on large network topology.

Cache size	FIFO Normal	FIFO Attack(pdet0,pat0.5)	FIFO Attack(pdet0.1,pat0.5)	FIFO Attack(pdet0.25,pat0.5)	FIFO Attack(pdet0.75,pat0.5)	FIFO Attack(pdet1,pat0.5)
1	1.7821	1.8255	1.7991	1.8001	1.7818	1.7968
5	1.2982	1.3505	1.3397	1.3455	1.3347	1.2875
10	0.8485	0.9505	0.9445	0.9137	0.8678	0.853
15	0.5616	0.6787	0.6618	0.6524	0.5891	0.5647
20	0.3613	0.4673	0.4554	0.4390	0.4021	0.3695
25	0.2372	0.3344	0.3180	0.3050	0.2725	0.2351

Table 4.16 Path length (hop count) values of normal situation versus attack situation with FIFO and pAttack 0.1

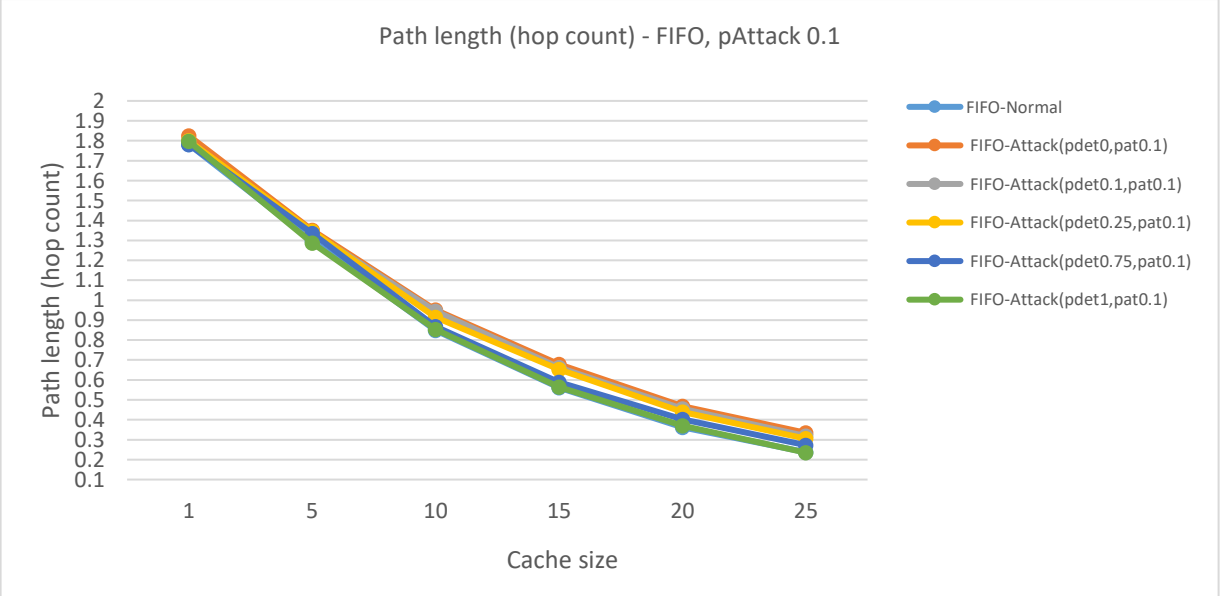

Figure: 4.16 Path length (hop count) – FIFO, pAttack 0.1

4.3. Evaluation

By evaluating the results and comparing our experiments it is obvious that attack has affected both small and large topologies. In small topology the variation of the results are not so high as we just have 3 nodes. However, the variation has increased in larger topology with 7 nodes. In a normal situation LRU is performing better than FIFO but when the system was under attack, LRU was affected a bit more than FIFO and as stated before, this is due to the different algorithms that they are using to replace the item in cache when cache is full. As cache size increases, we can see a reduction in both delay and path length in normal and attack situation. As a result, the highest percentage of increase in average delay in a small network topology during attack situation is 26.2 % and for average path length is 90 % which both has occurred on cache size 25 while the cache policy is LRU, pAttack is 0.5 and dAttack is 0. It is interesting to note that the highest amount of increase for average delay and average path length while pAttack 0.1 is again with same configuration as pAttack 0.5, but this time the percentage of increase for average delay is 14.5% and for path length is 47.7%. Therefore, pAttack 0.5 shows more impact on network than pAttack 0.1. Thus, when we have pAtack 0.5, we were able to get similar results to normal situation when dAttack is 1. However, with pAttack is 0.1, when dAttack is 0.75, the results are close to the normal experiment. It can be concluded that when we have 1% attack on the network a detection mechanism with even 75% accuracy is able to secure the network from attack.

Also when we have 50% attack possibility on our network, a mechanism with 1%, 25%

and even 75% are not sufficient to protect our system. In addition, in a Large network topology with 7 nodes, we have the highest percentage of increase again on cache size 25, LRU cache replacement policy, pAttack 0.5 and dAttack 0, where this time, the average delay has 29% increase and average path length has 67.4% percent increase . This results, demonstrate that the attack is even less effective on larger network than the small network. However, the average delay has increased a bit which is expected, but the differences between the average paths length, comparing to the small topology is 22.6% less, which is considerable. Moreover, when pAttack is 0.1 again the same configuration as pAttck 0.5, has the highest rate of increase which is 35.8% for average path length which is 11.9% less than the same condition in small network.

Chapter 5

Conclusion & Future Work

This thesis focuses on performance analysis of ICN cache services and we have simulated ICN cache pollution attacks to study the effectiveness of such attacks on ICN. In this final chapter we present conclusions about the work presented in this thesis and propose future directions for extending it. The first part summarizes the main contributions and draws conclusions. Then, we present potential directions for future investigations in the second part.

5.1 Conclusion

The growth of internet architecture and the central role that it plays in our society in terms of work and business, education, entertainment, social life and etc., brings the attention of researches to redesign and improve the weaknesses of current internet architecture. ICN is a new design that so many research studies has focused on it in recent years. Information centric networking is a new approach to evolve the internet infrastructure away from a host centric paradigm by replacing the IP based host to host networks with content based network. ICN has improved the network performance by caching the requested item through the path.

In this thesis we have studied the performance of ICN cache systems with and without

cache pollution attacks with different attack possibilities and attack detection probabilities. We have also studied LRU and FIFO cache replacement policies for each case on two different topologies. We calculated the delay and path length as our metrics to do the analysis. Our results show that attack was much more successful on larger cache sizes and LRU replacement policy has been affected more than FIFO during the attack. Although it was acting better during normal system. The results on the small and large network both display that the attack has much more negative impact on network while pAttack is 0.5. However, we cannot say that the attack was more successful on the larger topology as percentage of increase in larger topology is even less than small topology.

5.2 Future work

There are several areas of possible future work in the context of this thesis.

In this thesis we performed experimental analysis on testbed setup. However, we can implement this architecture in large and live network environment to check whether we are getting similar results or not.

Another possible extension can be repeating the experiments with another type of attack, in which the attacker has the knowledge of the contents stored in cache and knows about the cache sizes. This will be helpful to see in that scenario how much the attacker can leave an impact on network comparing to our current experiments. Furthermore, this work can be extended to use the results of our performance analysis, in designing a detection mechanism which is more effective and accurate to secure ICN cache services from cache attacks.

Bibliography:

[1] Xylomenos, George, et al. "A survey of information-centric networking research." IEEE Communications Surveys & Tutorials 16.2 (2014): 1024-1049.

[2] AbdAllah, Eslam G., Hossam S. Hassanein, and Mohammad Zulkernine. "A survey of security attacks in information-centric networking." IEEE Communications Surveys & Tutorials 17.3 (2015): 1441-1454.

[3] Deng, Leiwen, et al. "Pollution attacks and defenses for Internet caching systems." Computer Networks 52.5 (2008): 935-956.

[4] Guo, Haoran, et al. "Exploiting Path Diversity for Thwarting Pollution Attacks in Named Data Networking."

[5] Xie, Mengjun, Indra Widjaja, and Haining Wang. "Enhancing cache robustness for content-centric networking." INFOCOM, 2012 Proceedings IEEE. IEEE, 2012

[6] Gouge, Jeffery, Anand Seetharam, and Swapnoneel Roy. "On the scalability and effectiveness of a cache pollution based DoS attack in information centric networks." 2016 International Conference on Computing, Networking and Communications (ICNC). IEEE, 2016.

[7] Zhang, Lixia, et al. "Named data networking." ACM SIGCOMM Computer Communication Review 44.3 (2014): 66-73.

[8] Pan, Jianli, Subharthi Paul, and Raj Jain. "A survey of the research on future internet architectures." IEEE Communications Magazine 49.7 (2011): 26-36.

[9] Lal, Kumari Nidhi, and Anoj Kumar. "A Cache Content Replacement Scheme for Information Centric Network." Procedia Computer Science 89 (2016): 73-81.

[10] D. Trossen and G. Parisis, "Designing and realizing an information centric internet," IEEE Communications Magazine, Vol. 50, Issue 7, pp. 60-67, July, 2012.

[11] B. Mathieu, P. Truong, W. You, and J.-F. Peltier, "Information-centric networking: a natural design for social network applications," IEEE Communications Magazine, Vol. 50, Issue 7, pp. 44-51, July, 2012.

[12] Ho, Cheng-Yun, Cheng-Yuan Ho, and Chien-Chao Tseng. "A case study of cache performance in ICN—various combinations of transmission behavior and cache replacement mechanism." 2015 17th International Conference on Advanced Communication Technology (ICACT). IEEE, 2015.

[13] Saino, Lorenzo, Ioannis Psaras, and George Pavlou. "Icarus: a caching simulator for information centric networking (ICN)." Proceedings of the 7th International ICST Conference on Simulation Tools and Techniques. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2014.

[14] Lada A. Adamic1 Bernardo A. Huberman "Zipf's law and the Internet", Glottometrics 3, 2002,143-150

[15] Ahir, Deepali D., and Sagar B. Shinde. "Caching Simulators for Content Centric Networking." International Journal of Science and Research (IJSR) (2014).

[16] http://en.wikipedia.org/wiki/Zipf's_law

[17] P. Gasti, G. Tsudik, E. Uzun, and L. Zhang, "DoS & DDoS in named data networking", in Proc. of the 22nd International Conference on Computing Communications and Networks, IEEE, 2013.

[18] AbdAllah, Eslam G., Mohammad Zulkernine, and Hossam S. Hassanein. "Detection and Prevention of Malicious Requests in ICN Routing and Caching." Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing (CIT/IUCC/DASC/PICOM), 2015 IEEE International Conference on. IEEE, 2015.

[19] https://en.wikipedia.org/wiki/Gnutella

[20] Ghodsi, Ali, et al. "Information-centric networking: seeing the forest for the trees." Proceedings of the 10th ACM Workshop on Hot Topics in Networks. ACM, 2011.

[21] F. Almeida and J. Loureno, "Information centric networks-design issues, principles and approaches", International Journal of Latest Trends in Computing, vol. 3, no. 3, September 2012, pp. 58-66.

[22] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and Zipf-like Distributions: Evidence and Implications. In Proc. of INFOCOM, 1999.

[23] Y. Gao, L. Deng, A. Kuzmanovic, and Y. Chen, "Internet cache pollution attacks and countermeasures," in IEEE ICNP, 2006, pp. 54–64.

[24] V. Manivel, M. Ahamad, and H. Venkateswaran, "Attack resistant cache replacement for survivable services," in SSRS '03: Proceedings of the 2003 ACM workshop on Survivable and self-regenerative systems, 2003, pp. 64–71

[25] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs,and R. L. Braynard, "Networking named content," in ACM CoNEXT, 2009, pp. 1–12.

[26] E. J. Rosensweig, J. Kurose, and D. Towsley, "Approximate models for general cache networks," in INFOCOM, 2010

[27] Ghasemzadeh, Hassan, and Seyed Omid Fatemi. "Pseudo-FIFO Architecture of LRU Replacement Algorithm." 2005 Pakistan Section Multitopic Conference. IEEE, 2005.

[28] Hendrantoro, Gamantyo, and Achmad Affandi. "Early result from adaptive combination of LRU, LFU and FIFO to improve cache server performance in telecommunication network." Intelligent Technology and Its Applications (ISITIA), 2015 International Seminar on. IEEE, 2015.

[29] D. Lamet and J.F. Frenzel, Defect-Tolerant Cache Memory Design, IEEE Transactions on Computers, April 1993.